Homework 4: Convoluting Cats & Dogs

Your job is to create a convolutional neural network to distinguish images of cats and dogs. It is expected that you will use a neural network library such as TensorFlow, probably using the Keras library.

You will turn in three programs:

- Your neural network generator, which will be called `make_nn.py`. It will take two command-line arguments: a directory in which the images are located, and the name of the neural network file to create and save to disk. This program may assume that all training image file names begin with `c` or `d`, for "cat" and "dog" respectively.
- A classifier, called `classify.py`. Its first argument is the neural network to load up. All remaining arguments are image files to classify. It should print out one line for each file, with its name and whether it contains a cat or a dog. It may not make any assumptions about file names.
- Your best neural network. This should have a training accuracy in excess of 90%. Its name should be `name.dnn`, where `name` is your name. (You may use just your first name, if there's no one else in the class with your name.) Your network will be tested on an unseen testing set. *Be warned that this file will probably be quite large.*

Start by downloading the file `cats-and-dogs.zip`, which contains 25,000 separate 100×100-pixel JPEG images of cats and dogs. *Be aware that this file is over 100 MB long!* This file contains all the training data that you need. You may use all the data to train—don't worry about separating out a test set.

Keras will let you know the current accuracy of your network on the training set as it trains. Of course it will have an optimistic bias, because it is measuring accuracy on the training set itself. But the training set is large enough that the bias should be minimal.

You will have to come up with your own architecture for your network, including:

- How many layers there are.
- The numbers and sizes of kernels.
- The dropout rate to use.
- The sizes of your fully connected ("dense") layers.

However, your last layer will almost certainly be a "softmax" layer, that is trained on `[1,0]` for cats and `[0,1]` for dogs (or vice-versa). Remember that there is more than one right answer to this problem. If a model does not appear to be training well, stop early and try again with a different architecture.

When you save your neural network, don't save any optimizer information. In Keras, this can be done with:

```
model.save(model_file_name, include_optimizer = False)
```

Est 1888 UNIVERSITY of PUGET SOUND

This means that your saved model cannot be trained anymore, but it will also make your file sizes much more manageable.

Some hints:

- Training a model will take your computer *hours* to do. It's a good idea to let your computer do this overnight. Also, if you have a desktop computer at home, this may be faster than your laptop.
- You will likely need to install several Python modules. Even if you do this assignment at the last minute, please try to make the time to install the modules early, so that you know the program will work.
- Be careful of grayscale images, since they will not appear to have the same dimensionality as color images. (They will appear to be 100×100, not $100 \times 100 \times 1$ or $100 \times 100 \times 3$.) You will need to find a way to change them to be "color" images before training on them.
- Keras and TensorFlow assume that you will be working with NumPy arrays, rather than lists. It may be worth it to figure out how to use these before you venture into neural networks.
- Memory will be at a premium during this assignment. Most modern computers cope with memory limitations by using virtual memory: using swap space on the hard drive as if it were memory. This will have the effect of unreasonably slowing down your program. It may help you to monitor your computer's free memory while you are executing this code. Seemingly simple operations, like converting a list to an array, may be impossible to do in a reasonable amount of time.
- To save memory, consider using an economical data type ("`dtype`") for your NumPy arrays. Both `float16` and `uint8` might be very useful. There is absolutely no need for every pixel's data to need 3 separate `float64`s (AKA Java `double`s)!
- Beware of local minima while performing the gradient descent! If after a couple epochs, the accuracy is not improving, it is probably worth it to stop your program and try again. (Particularly annoying, some networks may get stuck classifying everything as a cat, or everything as a dog, and thus will *always* get 50% accuracy. Such models are not useful.)

Keep in mind that this assignment is relatively self-directed. Two different people might have very different networks, that still both get more than 95%.

Good luck!

UNIVERSITY of PUGET SOUND Est. 1888